# Assignment5

March 30, 2024

## 1 Assignment 5: Solutions

[17]:
```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import random as rand
```

### 1.1 1. Lotka-Voterra Model

**Consider the Lotka-Volterra Model of describing the number of prey $X(t)$ (e.g., hares) and predators $Y(t)$ (e.g., lynx) as described by the system of coupled differential equations with the harvesting of the prey and predators by the Hudson Bay Company. Here we consider a semi-sustainable model of harvest where individuals are removed in proportion to their density to the power $a$, $N^a$ where $a > 1$. This ensures that the rate of harvest drops quickly as the number of individuals declines**

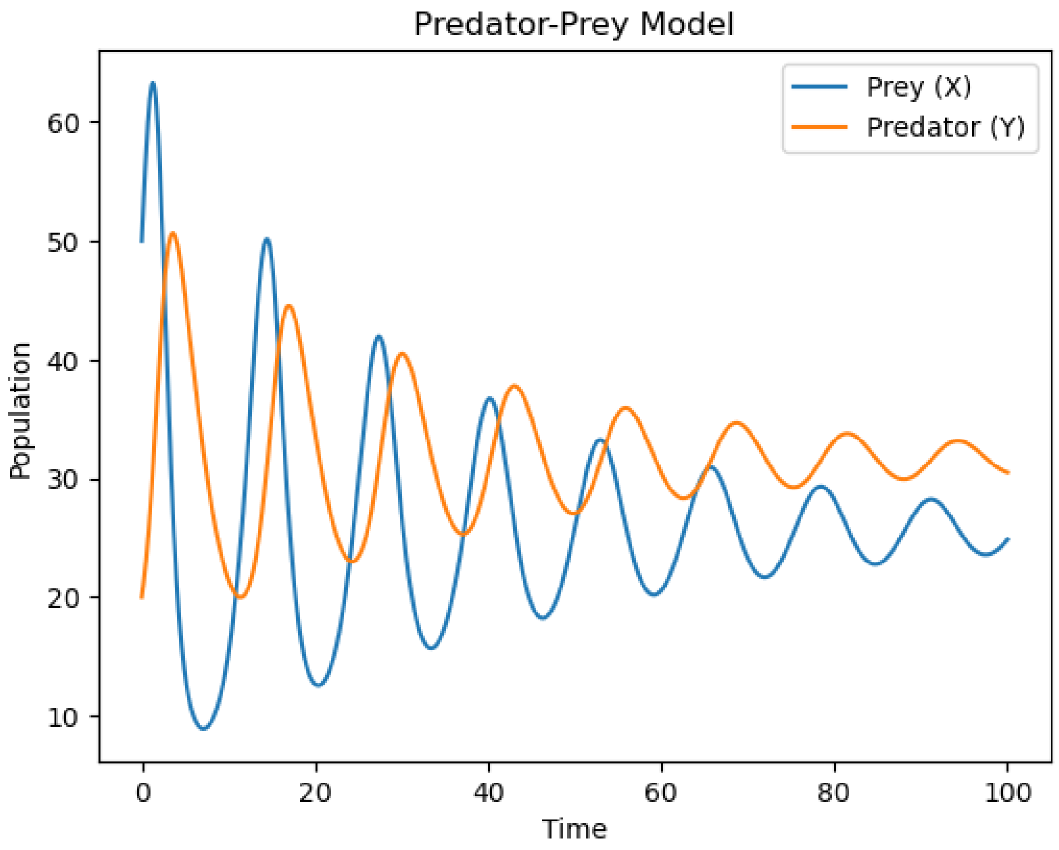
$$\frac{dX}{dt} = \underbrace{\alpha X}_{\text{Prey Birth}} - \underbrace{\beta XY}_{\text{Prey death}} - \underbrace{\mu_X X^{1.5}}_{\text{harvest}}$$
$$\frac{dY}{dt} = \underbrace{\delta XY}_{\text{Preditor birth}} - \underbrace{\gamma Y}_{\text{Preditor Birth}} - \underbrace{\mu_Y Y^{1.5}}_{\text{harvest}}$$

**Part A. Numerically integrate the ODEs above for $\alpha = 1$, $\beta = 0.03$, $\delta = 0.01$ and $\gamma = 0.2$ and $\mu_X = \mu = Y = 0.01$assuming we start with $X(0) = 500$ and $Y(0) = 200$.**

*Grading (2pt). Grade on completeion*

[56]:
```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Define the system of ODEs
def system(y, t, alpha, beta, delta, gamma, mu_X, mu_Y):
    X, Y = y
    dXdt = alpha * X - beta * X * Y - mu_X * X**1.5
    dYdt = delta * X * Y - gamma * Y - mu_Y * Y**1.5
    return [dXdt, dYdt]
```

```python
# Parameters
alpha = 1
beta = 0.03
delta = 0.01
gamma = 0.2
mu_X = 0.01
mu_Y = 0.01

# Initial conditions
initial_conditions = [50, 20]

# Time points for integration
t = np.linspace(0, 100, 1000)

# Numerical integration using odeint
solution = odeint(system, initial_conditions, t, args=(alpha, beta, delta,␣
 ↪gamma, mu_X, mu_Y))

# Extracting results
X, Y = solution.T

# Plotting
plt.plot(t, X, label='Prey (X)')
plt.plot(t, Y, label='Predator (Y)')
plt.xlabel('Time')
plt.ylabel('Population')
plt.legend()
plt.title('Predator-Prey Model')
plt.show()
```



**Part B. Propose an analogous continuous time stochastic process for this model. Describe the 2D state space of this model.**

*Grading (2pt). There is no one answer to this question, some may combine events that cause prey or predator death (e.g., death and harvest) this is okay.*

We consider a CTDS process with state vector $\vec{X} = [X, Y]$

| Event | Rate $\lambda_e$ | $\Delta_e \vec{X}$ |
|---|---|---|
| Prey Birth | $\alpha X$ | $[1, 0]$ |
| Predation | $\beta XY$ | $[-1, 0]$ |
| Prey Harvest | $\mu_x X$ | $[-1, 0]$ |
| Preditor Birth | $c\beta XY$ | $[0, 1]$ |
| Preditor Death | $\gamma Y$ | $[0, -1]$ |
| Preditor Harvest | $\mu_Y Y$ | $[0, -1]$ |

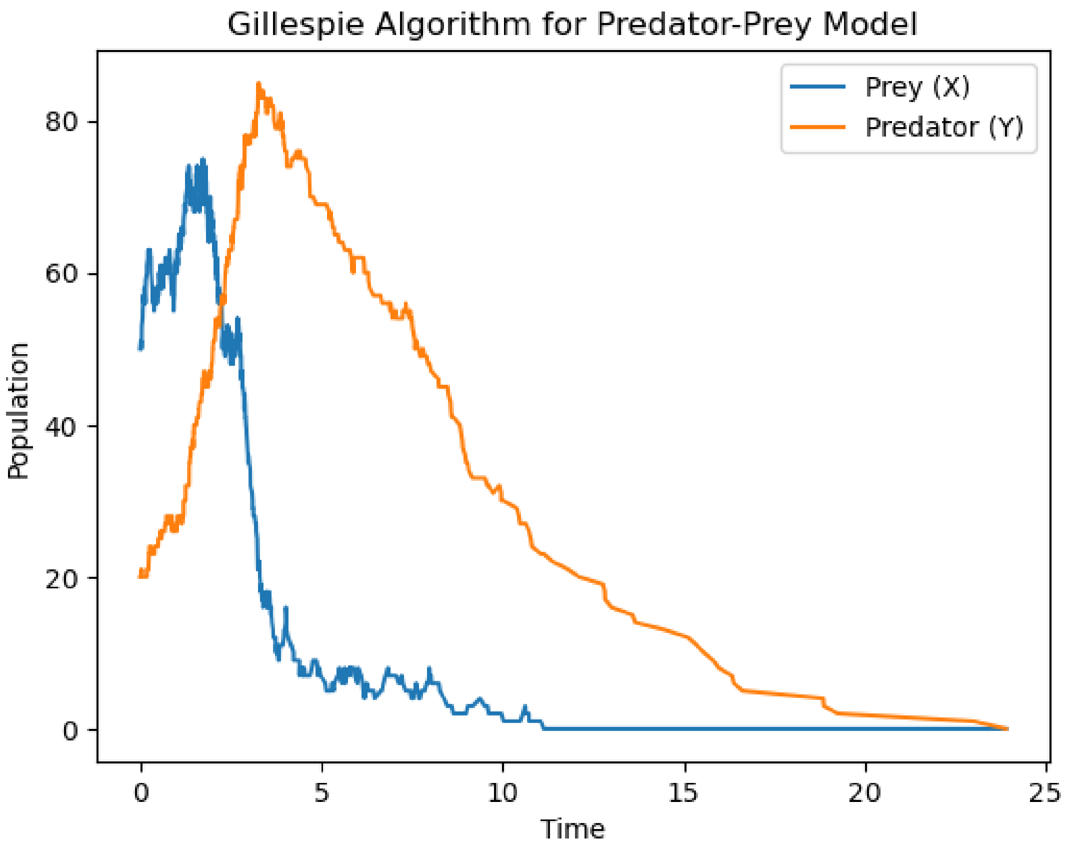
with parameters $\alpha = 2$, $\beta = 0.005$, $\delta = 0.001$ and $\gamma = 0.2$ and $\mu_X = \mu = Y = 0.01$assuming we start with $X(0) = 500$ and $Y(0) = 200$

**Part C. Simulate the dynamics of the stochastic model you proposed and compare the simulated trajectories to your answer in part 1.**

*Grading (2pt). Spot-check the code and look at the plot.*

[55]:
```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
alpha = 1
beta = 0.03
delta = 0.01
gamma = 0.2
mu_X = 0.01
mu_Y = 0.01

def sim():
    # Initial conditions
    X = 50
    Y = 20

    # Simulation time
    t_max = 50
    t = 0

    # Lists to store results
    times = [t]
    populations_X = [X]
    populations_Y = [Y]

    rates = [alpha * X, beta * X * Y, mu_X * X, delta * X * Y, gamma * Y, mu_Y␣
 ↪* Y]
    total_rate = sum(rates)
    dt = -np.log(np.random.uniform()) / total_rate #This is another way of␣
 ↪getting a random exponentially distributed #

    # Gillespie algorithm
    while t+dt < t_max and total_rate>0:

        # Choose the event
        event = np.random.choice(range(6), p=[rate / total_rate for rate in␣
 ↪rates])

        # Update state vector
        X += [1,-1, -1, 0, 0, 0][event]
        Y += [0, 0, 0, 1, -1, -1][event]

        # Update time
        t += dt
        # Store results
        times.append(t)
        populations_X.append(X)
        populations_Y.append(Y)

        # Calculate rates
        rates = [alpha * X, beta * X * Y, mu_X * X, delta * X * Y, gamma * Y,␣
 ↪mu_Y * Y]

        # Calculate total rate
        total_rate = sum(rates)

        # Calculate time to next event
        dt = -np.log(np.random.uniform()) / total_rate
    return [times,populations_X,populations_Y]

test=sim()
# Plotting
plt.plot(test[0], test[1], label='Prey (X)')
plt.plot(test[0], test[2], label='Predator (Y)')
plt.xlabel('Time')
plt.ylabel('Population')
plt.legend()
plt.title('Gillespie Algorithm for Predator-Prey Model')
plt.show()
```

```
/tmp/ipykernel_214/2338143238.py:55: RuntimeWarning: divide by zero encountered
in scalar divide
  dt = -np.log(np.random.uniform()) / total_rate
```



**Part D: Derive a system of master equations for the probability of having $n$ prey and $m$ predators at time $t$ in the absence of harvesting $\mu_X = \mu_Y = 0$. You do NOT need to solve them numerically.**

*Grading (4pt). Grade for correctness.*

Let $P_{n,m}$ be the probability that a population has $n$ prey and $m$ predators at time $t$.

$$\begin{aligned}\frac{dP_{n,m}}{dt} = &-(\alpha n + \beta nm + \delta nm + \gamma m)\, P_{n,m} \quad \text{nothing happens}\\ &+\alpha(n-1)P_{n-1,m} + \delta n(m-1)P_{n,m-1} \quad \text{prey and pred birth}\\ &+\beta(n+1)mP_{n+1,m} + \gamma(m+1)P_{n,m+1} \quad \text{prey and pred death}\end{aligned}$$

### 1.2 2. SIS Model

**Consider the model given in example 5.13: a stochastic SIS model where transmissions occur at a mass-action rate of $\frac{\beta}{\kappa} * S * I$ with $\beta = 0.5$, hosts recover (becoming susceptible again) at a rate $\gamma = 0.1$ and the total population size is $\kappa = 100$.**

**Our goal here is to model the mean and variance in the number of infections in this model using an ensemble moment approximation.**

**Part A. What are the ODEs that describe the dynamics of $\langle I \rangle$ and $\langle I^2 \rangle$? How do these equations depend on $\langle I^3 \rangle$ and why?**

*Grading (4pt). Check initial equations, especially for correctness.*

Let's start by writing a table of events, rates, and effects:

| Event | Rate $\lambda_e$ | Effect |
|---|---|---|
| Transmission | $\frac{\beta}{\kappa}(\kappa - I)I$ | $[-1, 1]$ |
| Recover | $\gamma I$ | $[1, -1]$ |

$$\begin{aligned}\frac{d\langle I \rangle}{dt} &= \left\langle \frac{\beta}{\kappa}(\kappa - I)I \times (I + 1 - I) + \gamma I \times (I - 1 - I) \right\rangle \\ &= \beta \langle I \rangle - \frac{\beta}{\kappa} \langle I^2 \rangle - \gamma \langle I \rangle\end{aligned}$$

and

$$\begin{aligned}\frac{d\langle I^2 \rangle}{dt} &= \left\langle \frac{\beta}{\kappa}(\kappa - I)I \times \underbrace{((I+1)^2 - I^2)}_{1+2I} + \gamma I \times \underbrace{((I-1)^2 - I^2)}_{1-2I} \right\rangle \\ &= (\beta + \gamma)\langle I \rangle + \left(\beta\left(2 - \frac{1}{\kappa}\right) - 2\gamma\right)\langle I^2 \rangle - \frac{2\beta}{\kappa}\langle I^3 \rangle\end{aligned}$$

**Part B. Implement moment closure assuming that the skew in the number of infections is small.**

*Grading (3pt). Spot Check.*

$Skew(I) = \langle I^3 \rangle - 3\langle I^2 \rangle \langle I \rangle + 2\langle I \rangle^3 \approx 0$

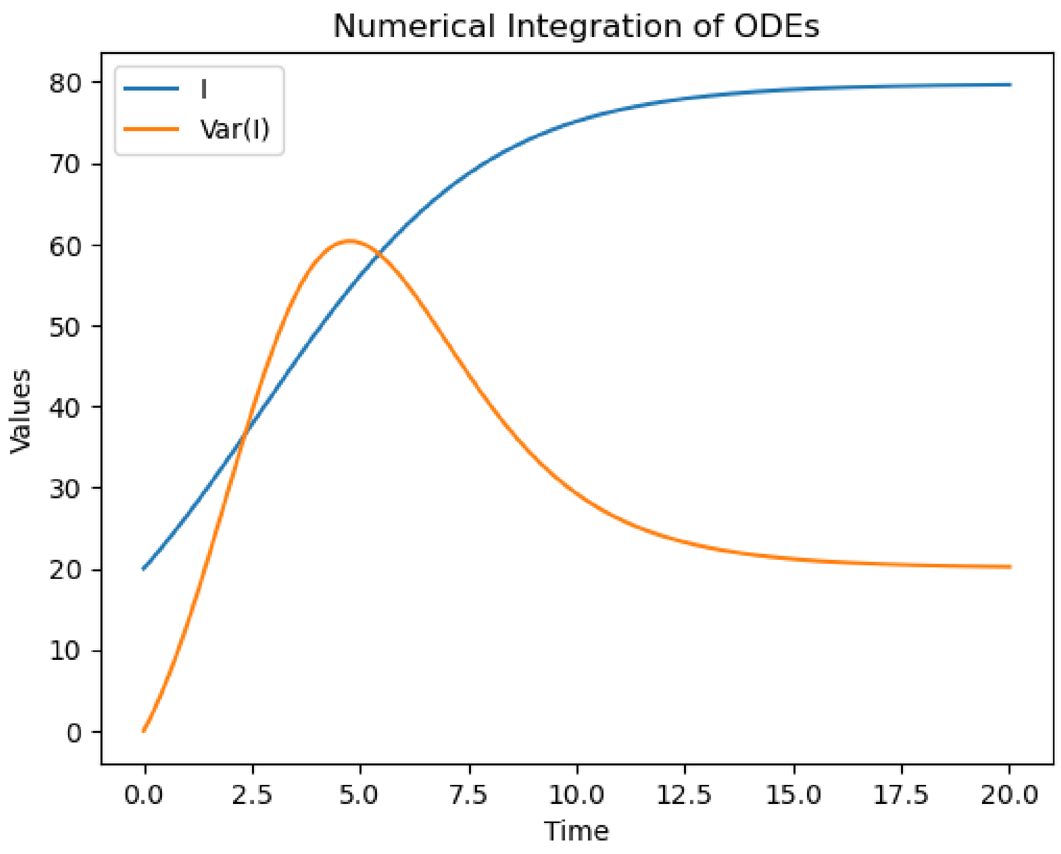
Hence $\langle I^3 \rangle \approx 3\langle I^2 \rangle \langle I \rangle - 2\langle I \rangle^3$

$$\begin{aligned}\frac{d\langle I \rangle}{dt} &= \beta \langle I \rangle - \frac{\beta}{\kappa}\langle I^2 \rangle - \gamma \langle I \rangle \\ \frac{d\langle I^2 \rangle}{dt} &= (\beta + \gamma)\langle I \rangle + \left(\beta\left(2 - \frac{1}{\kappa}\right) - 2\gamma\right)\langle I^2 \rangle - \frac{2\beta}{\kappa}\left(3\langle I^2 \rangle \langle I \rangle - 2\langle I \rangle^3\right)\end{aligned}$$

**Part C. Numerically solve the dynamics for the mean and variance assuming $\beta = 0.5, \gamma = 0.1, \kappa = 100$ and $I(0) = 20$ and no variance for the first 10 units of time.**

*Grading (3pt). Grade by completion*

[61]:
```python
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Define the system of ODEs
def system(y, t, beta, gamma, kappa):
    I, I_squared = y
    dIdt = beta * I - (beta / kappa) * I_squared - gamma * I
    dI_squared_dt = (beta + gamma) * I + (beta * (2 - 1 / kappa) - 2 * gamma) *␣
 ↪I_squared - (2 * beta / kappa) * (3 * I_squared * I - 2 * I**3)
    return [dIdt, dI_squared_dt]

# Parameters
beta = 0.5
gamma = 0.1
kappa = 100

# Initial conditions
initial_conditions = [20, 20**2]

# Time points for integration
t = np.linspace(0, 20, 1000)

# Numerical integration using odeint
solution = odeint(system, initial_conditions, t, args=(beta, gamma, kappa))

# Extracting results
I, I_squared = solution.T

# Plotting
plt.plot(t, I, label='I')
plt.plot(t, I_squared-I**2, label='Var(I)')
plt.xlabel('Time')
plt.ylabel('Values')
plt.legend()
plt.title('Numerical Integration of ODEs')
plt.show()
```



**Part D. Challenge 795: Using your numerical solution in 3 comment on whether your approximation is valid for the time period considered.**

*Grading (2pt). Completion w/ good reasoning.*

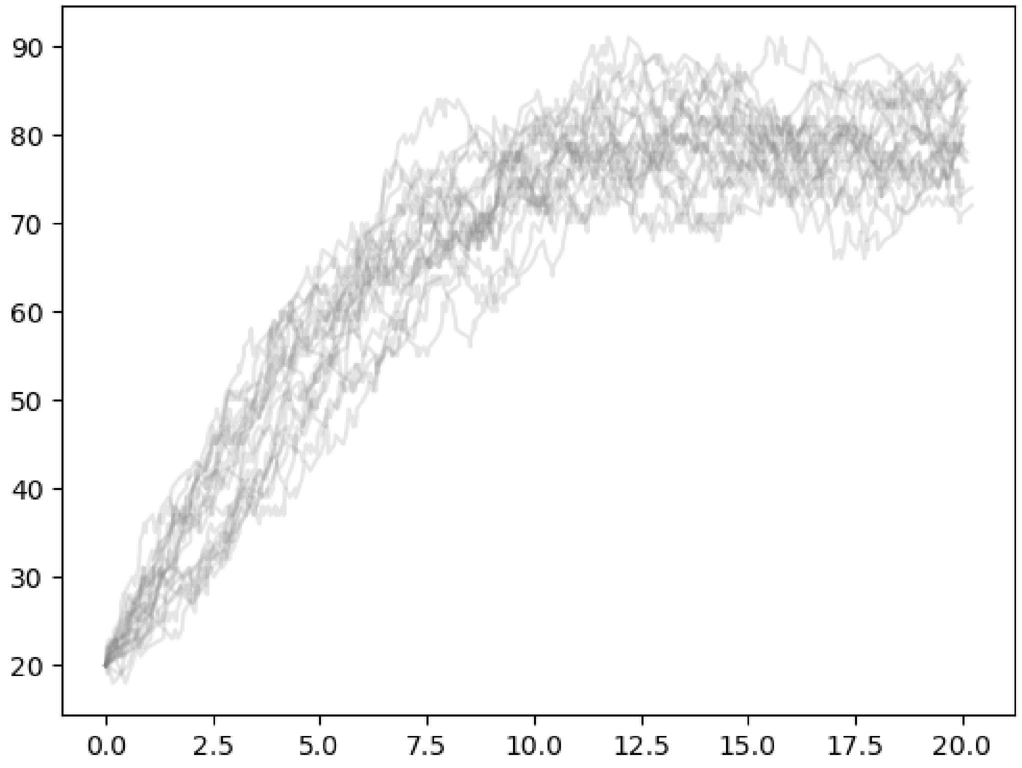
The variance is always on a similar scale as the the mean so we are okay!

**Part E. Challenge 795: Compare your solution to the EMA to the result you obtain from a simulation approach.**

*Grading (3pt). Grade by inspecting the plot. There are several ways to "compare" the results so as long as the plot captures some sort of comparison that is okay.*

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
beta = 0.5
gamma = 0.1
kappa = 100

def SIS_Sim():
    # Initial conditions
    S = 80
    I = 20

    # Simulation time
    t_max = 20
    t = 0

    # Lists to store results
    times = [t]
    susceptible_populations = [S]
    infected_populations = [I]

    # Gillespie algorithm
    while t < t_max:
        # Calculate rates
        transmission_rate = (beta / kappa) * (kappa - I) * I
        recover_rate = gamma * I
        total_rate = transmission_rate + recover_rate

        # Calculate time to next event
        dt = -np.log(np.random.uniform()) / total_rate

        # Choose the event
        event = np.random.choice([0, 1], p=[transmission_rate / total_rate,␣
 ↪recover_rate / total_rate])

        # Update state vector
        S += [-1, 1][event]
        I += [1, -1][event]

        # Update time
        t += dt

        # Store results
        times.append(t)
        susceptible_populations.append(S)
        infected_populations.append(I)
    return [times,susceptible_populations,infected_populations]
test=SIS_Sim();
# Plotting
for i in range(20):
    test=SIS_Sim();
    plt.plot(test[0], test[2],color='gray',alpha=0.2)
plt.plot(t, I,color='blue')
plt.plot(t, I+2*np.sqrt(I_squared-I**2),color='blue',alpha=0.5)
plt.plot(t, I-2*np.sqrt(I_squared-I**2),color='blue',alpha=0.5)
plt.xlabel('Time')
plt.ylabel('Population')
plt.title('Gillespie Algorithm for SIS Model')
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 53
     51     test=SIS_Sim();
     52     plt.plot(test[0], test[2],color='gray',alpha=0.2)
---> 53 plt.plot(t, I,color='blue')
     54 plt.plot(t, I+2*np.sqrt(I_squared-I**2),color='blue',alpha=0.5)
     55 plt.plot(t, I-2*np.sqrt(I_squared-I**2),color='blue',alpha=0.5)

NameError: name 't' is not defined
```



The simulations fit squarely within the $\pm 2SD$ interval.

## 1.3 3. The Yule Process

**In the lecture we derived the master equation for the size of a clade in the Yule model:**

$$\frac{dP_n(t)}{dt} = -\lambda n P_n + \lambda(n-1)P_{n-1} \quad P_n(0) = \begin{cases} 0 & n \neq 1 \\ 1 & n = 1 \end{cases}$$

**with the solution:**

$$P(n,t) = e^{-n\lambda t}\left(e^{\lambda t} - 1\right)^{(n-1)}$$

**Part A. Challenge for 795: Show that this is the case** *Grading (2pt). Quick check*

There are two approaches to showing that this is the case. One is to integrate the system of ODEs the other is to differentiate the solution.

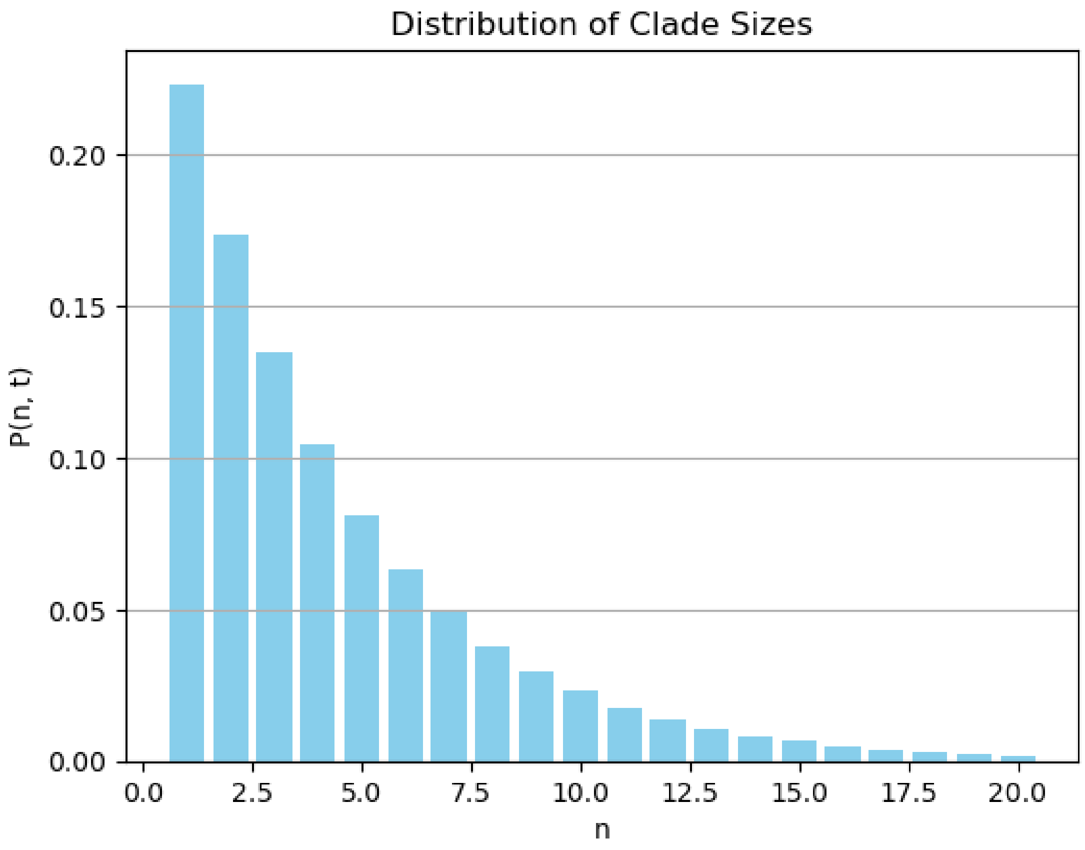
Taking the derivative approach:

$$\frac{d}{dt}P_{n,t} = \lambda(n-1)\underbrace{e^{-\lambda(n-1)t}\left(e^{\lambda t}-1\right)^{n-2}}_{P_{n-1}(t)} - \lambda n\underbrace{e^{-\lambda n t}\left(e^{\lambda t}-1\right)^{n-1}}_{P_n(t)}$$

**Part B: Plot the distribution of clade sizes at $\lambda = 1$ at $T = 1.5$ for $n = 1 \ldots 20$**

*Grading (3pt).: Grade by inspecting plot.*

[7]:
```python
# Function definition
def P(n, t, lmbda):
    return np.exp(-n * lmbda * t) * (np.exp(lmbda * t) - 1)**(n - 1)

# Parameters
lmbda = 1
t = 1.5
n_values = np.arange(1, 21)

# Calculate P values
P_values = [P(n, t, lmbda) for n in n_values]

# Plotting as bar plot
plt.bar(n_values, P_values, color='skyblue')
plt.title(f'Distribution of Clade Sizes')
plt.xlabel('n')
plt.ylabel('P(n, t)')
plt.grid(axis='y')
plt.show()
```



**Part C: Technically the number of lineages at time $T$ could be infinite, obviously we can make a plot in part 2 with an infinite domain. How close is the approximation of the probability distribution you showed in part 2 to the truth? In other words how much of the total probability are you missing by showing a plot on a finite domain?**

*Grading (4pt). Calculates "missing probability"*

[12]:
```
temp=np.array([P(n, t, lmbda) for n in range(1,20)])
print(temp)

# Missing probability: 1-sum(P(n,t))
print(1-np.sum(temp))
```

```
[0.22313016 0.17334309 0.13466502 0.10461719 0.08127394 0.06313927
 0.049051   0.03810624 0.02960359 0.02299814 0.01786656 0.01387999
 0.01078295 0.00837695 0.0065078  0.00505571 0.00392763 0.00305126
 0.00237043]
0.008253097293440925
```

So awe are missing 0.8%

**Part D: Simulate 50 trees in the Yule model for these parameters. Do your simulations**

**match the solutions from the master equation?**

*Grading (3pt). Grade by visual inspection of plot or output of code.*

[30]:
```
def SimYule(lam,tMax):
    #initialization
    t=0;
    C=np.array([[0]]);
    n=C.shape[0]
    rateTot=lam*n
    Deltat=np.random.exponential(scale=1/rateTot);
    t=t+Deltat
    while t<tMax:
        C=C+np.identity(n)*Deltat #Update time
        par=rand.random_integer = rand.randint(0, n - 1) #Choose parent
        C=np.vstack([C, C[par]]) # Add parental row
        C=np.hstack([C,C[:,par].reshape(-1, 1)]) # Add parental column
        n=C.shape[0] # Update n
        rateTot=lam*n # Calculate new rate
        Deltat=np.random.exponential(scale=1/rateTot); # Choose next branching␣
↪time
        t=t+Deltat
    C=C+np.identity(n)*Deltat #Add terminal branches
    return C
```

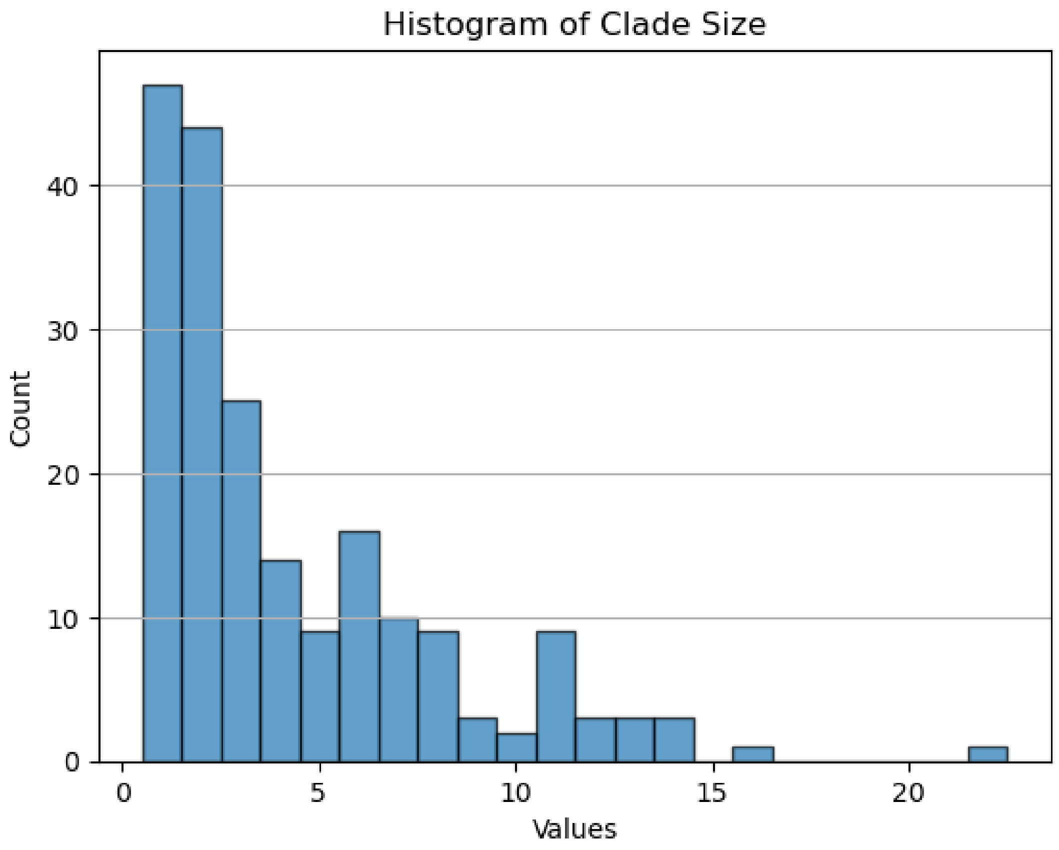
We can easily calculate the size of one simulation

[31]:
```
len(SimYule(1,1.5))
```

[31]:
```
34
```

[32]:
```
# Dictionary to store simulation results
simulations = {}

# Perform 50 simulations
for index in range(200):
    simulations[index]=SimYule(1,1.5)
```

[36]:
```
sizeList=np.array([len(simulations[index]) for index in range(200)])
# Plotting histogram
bin_width = 1
bins = np.arange(min(sizeList), max(sizeList) + bin_width, bin_width) - 0.5
plt.hist(sizeList, bins=bins, edgecolor='black', alpha=0.7)
plt.title('Histogram of Clade Size')
plt.xlabel('Values')
plt.ylabel('Count')
plt.grid(axis='y')
plt.show()
```



Looks good!

### 1.4 4. The Birth-Death Process

**Part A: Propose a system of Master Equations describing the size of a clade in the birth-death model.**

*Grading (4pt): Check*

Let $P_n(t)$ be the probability that a clade has $n$ lineages at time $t$.

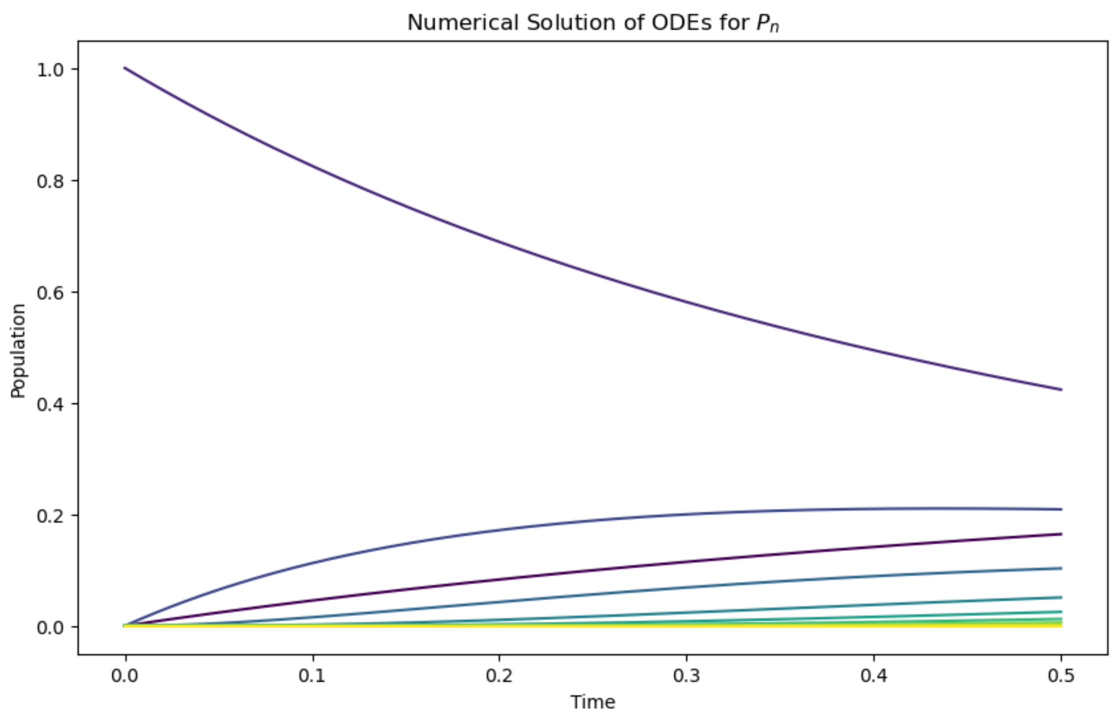
$$\frac{dP_n(t)}{dt} = -(\lambda n + \mu n)P_n(t) + \lambda(n-1)P_{n-1}(t) - \mu(n+1)P_{n+1}(t)$$

**Part B: Numerically solve these equations assuming $\lambda = 1.5$ and $\mu = 0.5$ and for $t < T = 0.5$.**

*Grading (3pt): Completion*

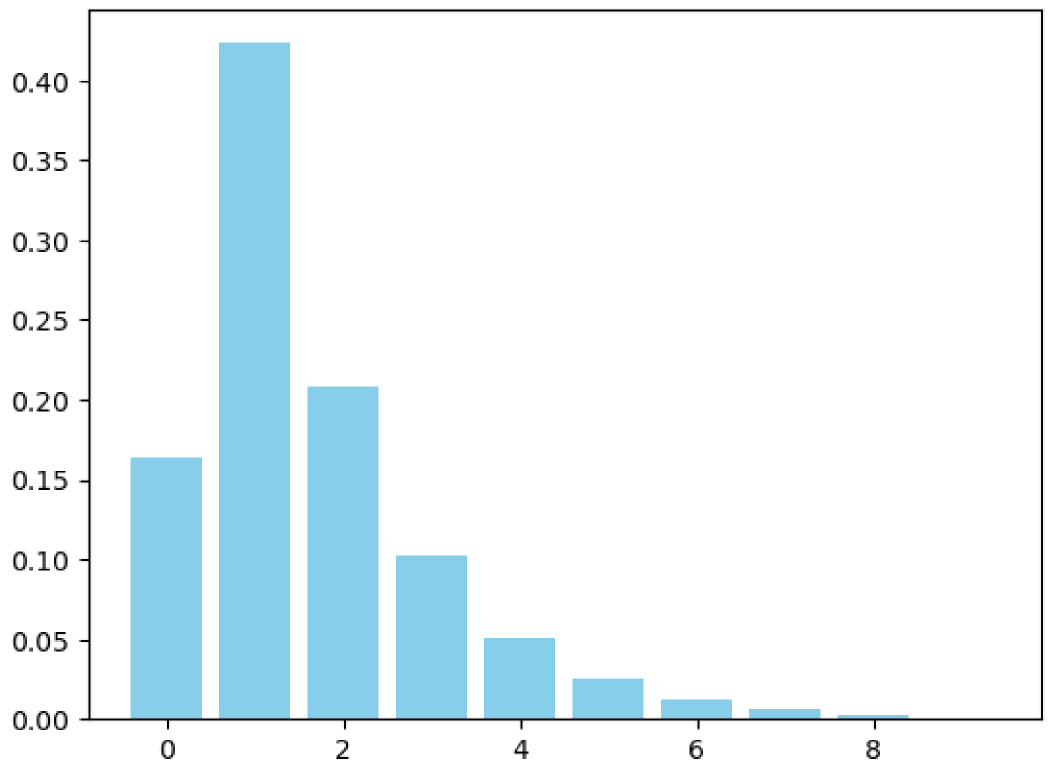
[183]:
```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
```

```
# Define the system of ODEs
def system(P, t, lambda_val, mu_val):
    N = len(P)
    dPdt = np.zeros(N)
    dPdt[0] = mu_val * P[1]
    for n in range(1, N-1):
        if n==N-1:
            dPdt[n] = - (lambda_val * n + mu_val * n) * P[n] + lambda_val * (n␣
 ↪- 1) * P[n - 1]
        else:
            dPdt[n] = - (lambda_val * n + mu_val * n) * P[n] + lambda_val * (n␣
 ↪- 1) * P[n - 1] + mu_val * (n + 1) * P[n + 1]
    return dPdt

# Parameters
lambda_val = 1.5
mu_val = 0.5

# Initial conditions
nMax=10
P0 = np.zeros(nMax)
P0[1] = 1

# Time points for integration
t = np.linspace(0, 0.5, 1000)

# Numerical integration using odeint
solution = odeint(system, P0, t, args=(lambda_val, mu_val))

# Extracting results
P_n = solution.T

# Plotting
colors = plt.cm.viridis(np.linspace(0, 1, len(P0)))
plt.figure(figsize=(10, 6))
for n in range(0,nMax):
    plt.plot(t, P_n[n], color=colors[n])

plt.xlabel('Time')
plt.ylabel('Population')
plt.title('Numerical Solution of ODEs for $P_n$')
plt.show()
```



```
[184]: dist=np.array([P_n[n][-1] for n in range(nMax)]);
       size=np.array([n for n in range(nMax)])
       plt.bar(size, dist, color='skyblue')
```

```
[184]: <BarContainer object of 10 artists>
```



At the time plotted the distribution only captures 99.8% of the probability density

[185]:
```
dist=np.array([P_n[n][-1] for n in range(nMax)]);
np.sum(dist)
```

[185]:
```
0.9967516666159619
```

**Part C: How does clade size in this model compare to the Yule model above? Are you surprised they are the same/different?**

*Grading (3pt): Completion*

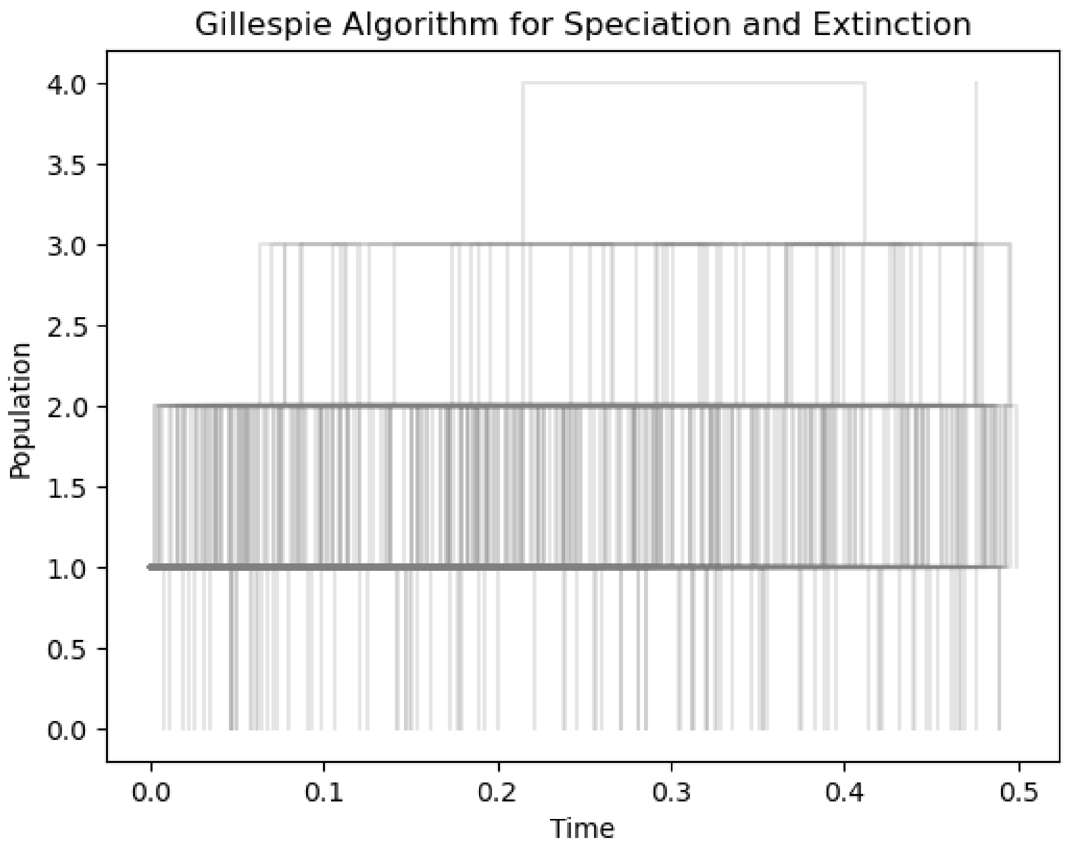
Let's start by writing a table of events, rates, and effects:

| Event | Rate $\lambda_e$ | Effect |
|---|---|---|
| Speciation | $\lambda n$ | $[1]$ |
| Extinction | $\mu n$ | $[-1]$ |

[201]:
```
import numpy as np
import matplotlib.pyplot as plt

def gillespie_algorithm(initial_population, lambda_val, mu_val, max_time):
    # Initialize time and populations
    t = 0
    n = initial_population
    times = [t]
    populations = [n]
    rates = [mu_val * n, lambda_val * n]
    total_rate = sum(rates)
    dt = -np.log(np.random.uniform()) / total_rate
    while t+dt < max_time and total_rate>0:
        # Choose the next event
        event = np.random.choice([0,1], p=[rates[0] / total_rate, rates[1] /␣
 ↪total_rate])

        # Update time and population
        t += dt
        n += [-1, 1][event]

        # Store results
        times.append(t)
        populations.append(n)

        # Calculate rates for each event
        rates = [lambda_val * n, mu_val * n]

        # Calculate total rate
        total_rate = sum(rates)

        dt = -np.log(np.random.uniform()) / total_rate

    return times, populations

# Parameters
lambda_val = 1.5
mu_val = 0.5
initial_population = 1
max_time = 0.5

# Run the Gillespie algorithm
times, populations = gillespie_algorithm(initial_population, lambda_val,␣
 ↪mu_val, max_time)

nFinal=np.array([])
# Plotting
for i in range (500):
    times, populations = gillespie_algorithm(initial_population, lambda_val,␣
 ↪mu_val, 0.5)
    plt.step(times, populations, where='post',color='gray',alpha=0.2)
    nFinal=np.append(nFinal,populations[-1])
```

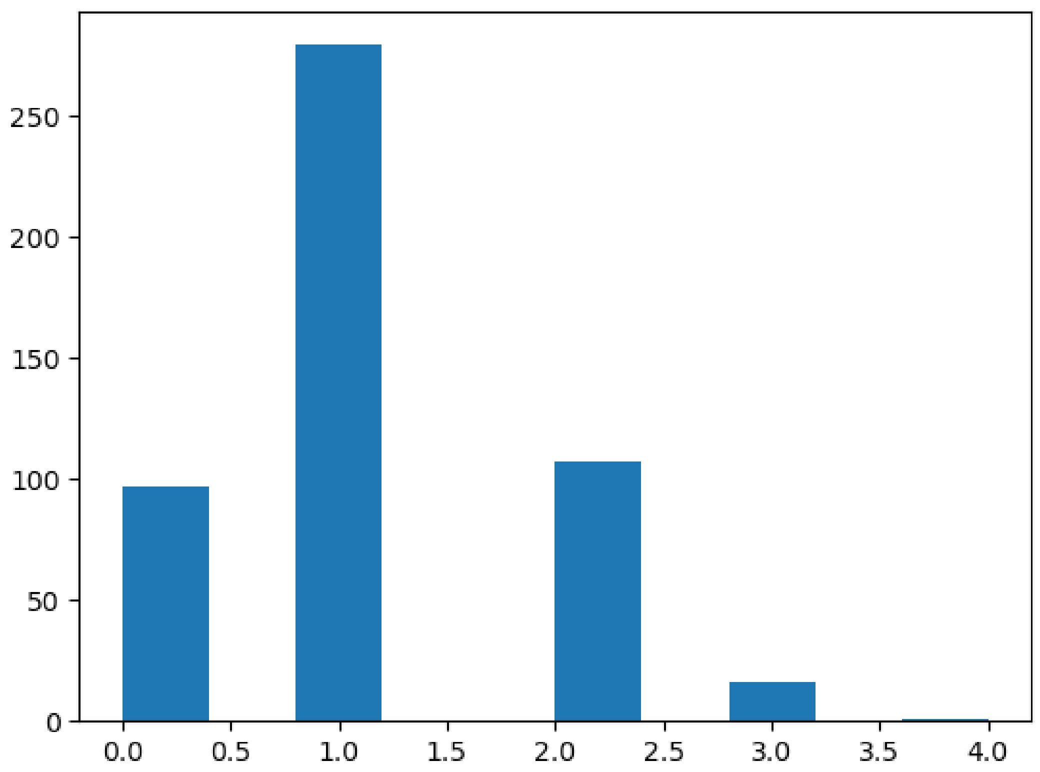
```
plt.xlabel('Time')
plt.ylabel('Population')
plt.title('Gillespie Algorithm for Speciation and Extinction')
plt.show()
```

```
/tmp/ipykernel_214/4038247896.py:31: RuntimeWarning: divide by zero encountered
in scalar divide
  dt = -np.log(np.random.uniform()) / total_rate
```



[202]:
```
plt.hist(nFinal)
```

[202]:
```
(array([ 97.,   0., 279.,   0.,   0., 107.,   0.,  16.,   0.,   1.]),
 array([0. , 0.4, 0.8, 1.2, 1.6, 2. , 2.4, 2.8, 3.2, 3.6, 4. ]),
 <BarContainer object of 10 artists>)
```


250
200
150
100
50
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0